CS420: Operating Systems

Paging and Page Tables

James Moscola
Department of Physical Sciences
York College of Pennsylvania

Paging

- **Paging is a memory-management scheme that permits the physical address space of a process to be noncontiguous**
  - Avoids external fragmentation
  - Avoids the need for compaction
  - May still have internal fragmentation
- **Divide physical memory into fixed-sized blocks called frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
  - Must keep track of all free frames
- **Divide logical memory into blocks of same size called pages**
  - Backing store is also split into pages of the same size
- **To run a program of size *N* pages, need to find *N* free frames and load program**

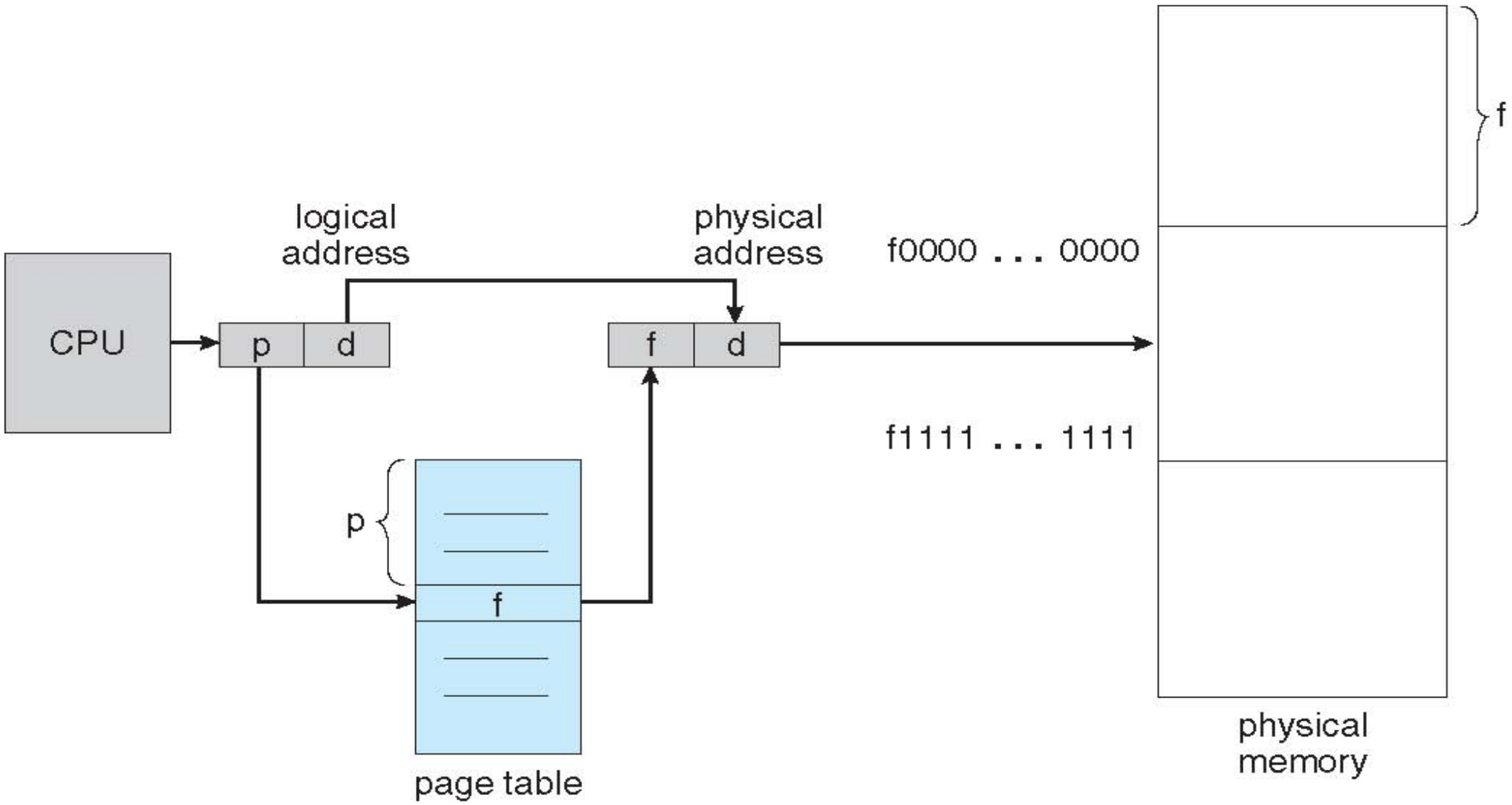
Address Translation Scheme

- **Set up a page table to translate logical to physical addresses**
- **Address generated by CPU is divided into:**
  - Page number (p) – used as an index into a page table which contains base address of each page in physical memory
  - Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit

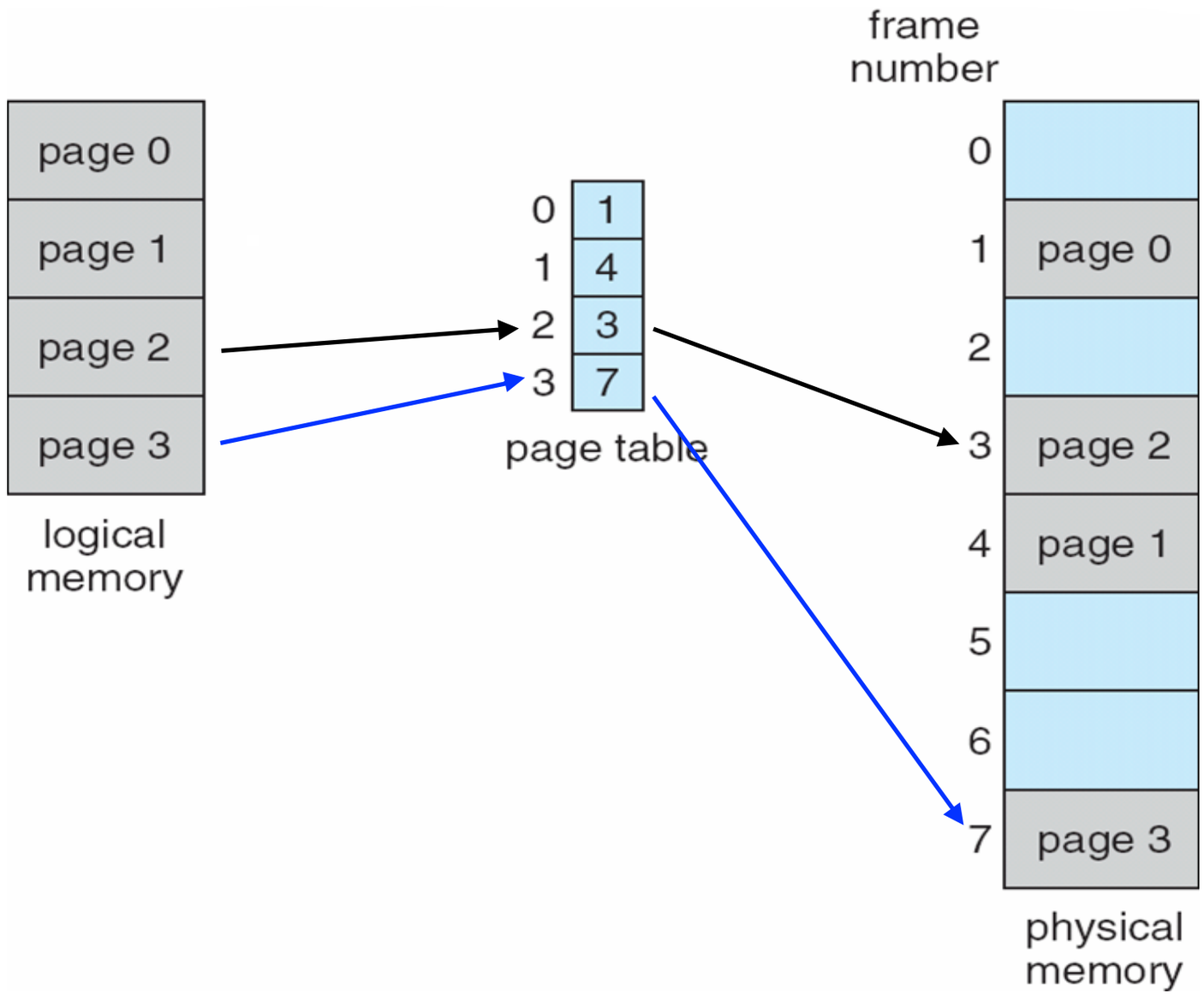
- For given logical address space 2^m and page size 2^n

page number	page offset
p	*d*
m - n	*n*

Paging Hardware

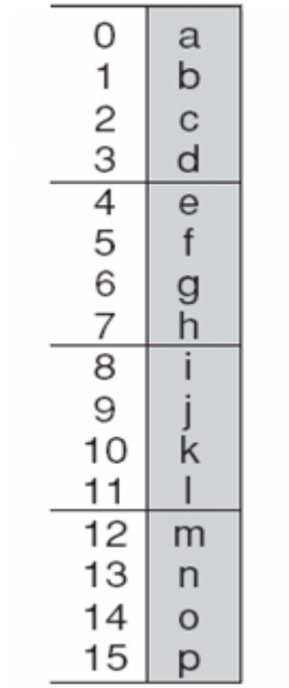


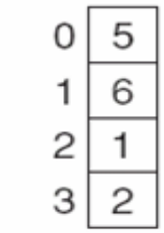
Paging Model of Logical and Physical Memory

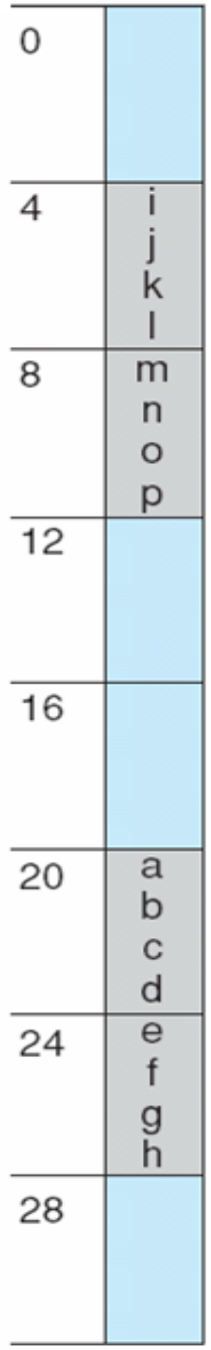


Paging Example

page number	page offset
p	*d*
m - n	*n*



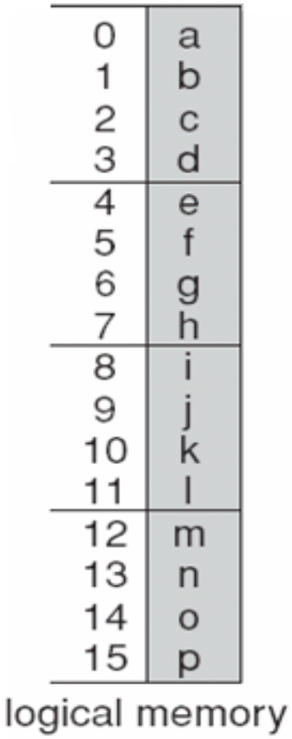




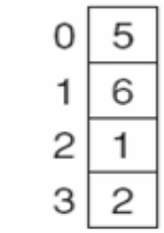
- **Example setup:**
  - n=2 and m=4 (*p* and *d* are each 2-bits)
  - 32-bytes of physical memory and
  - 4-byte pages (8 pages can fit in physical memory space)
- **Example:**
  - Logical address 10 is in page 2 at offset 2
  - According to the page table, page 2 is located in frame 1
  - Physical memory address is: (Frame # * Page size) + Offset
  - Physical memory address for logical address 10 is : (1 * 4 bytes) + 2 byte offset = 6

Another Paging Example

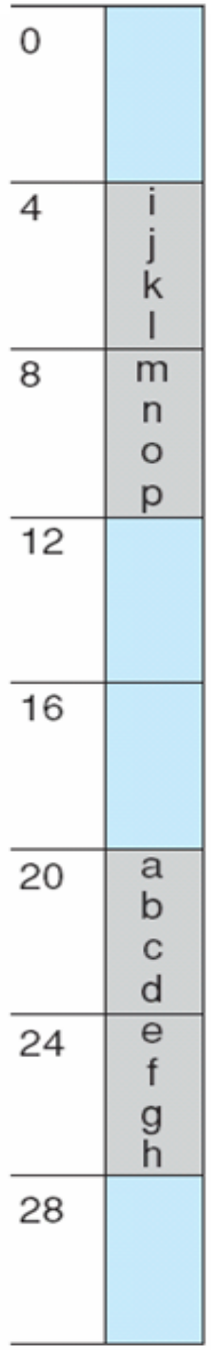
page number	page offset
p	*d*
m - n	*n*



logical memory



page table

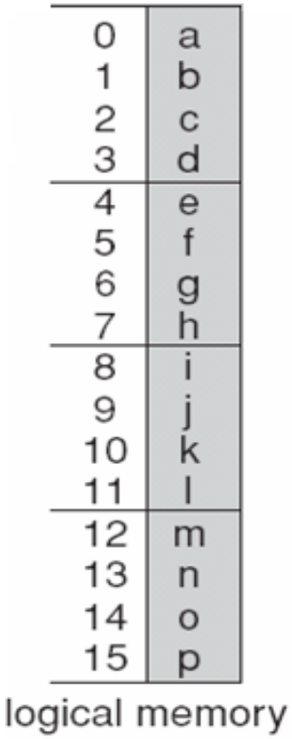


physical memory

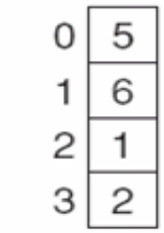
- **Example setup:**
  - n=2 and m=4 (*p* and *d* are each 2-bits)
  - 32-bytes of physical memory and
  - 4-byte pages (8 pages can fit in physical memory space)
- **Example:**
  - Logical address 4 is in page 1 at offset 0
  - According to the page table, page 1 is located in frame 6
  - Physical memory address is: (Frame # * Page size) + Offset
  - Physical memory address for logical address 10 is : (6 * 4 bytes) + 0 byte offset = 24

Another Paging Example

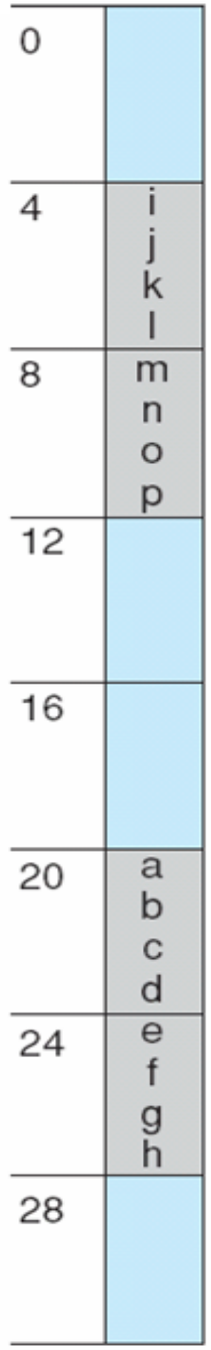
page number	page offset
p	*d*
m - n	*n*



logical memory



page table



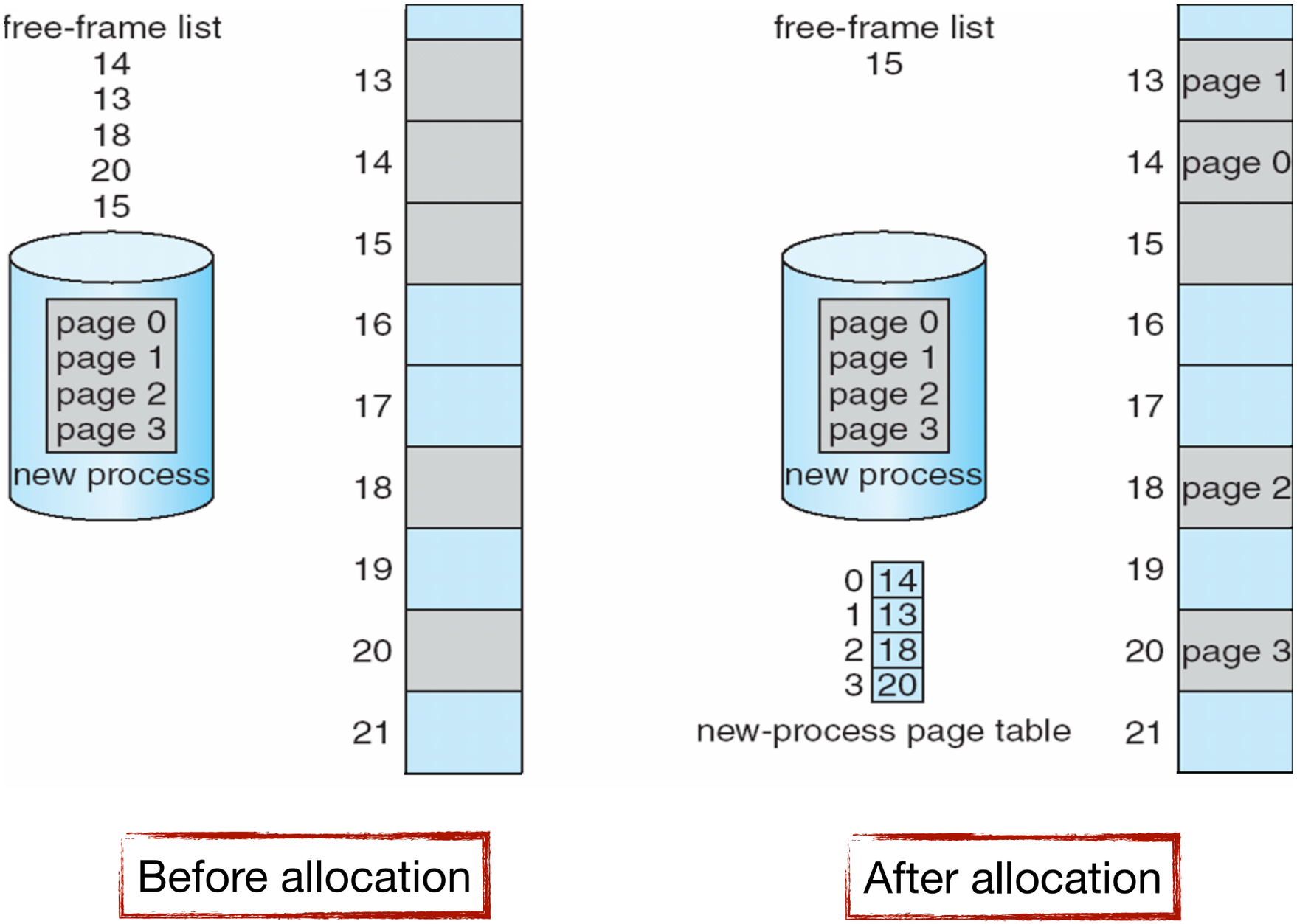
physical memory

- **Example setup:**
  - n=2 and m=4 (*p* and *d* are each 2-bits)
  - 32-bytes of physical memory and
  - 4-byte pages (8 pages can fit in physical memory space)
- **Example:**
  - Logical address 7 is in page 1 at offset 3
  - According to the page table, page 1 is located in frame 6
  - Physical memory address is: (Frame # * Page size) + Offset
  - Physical memory address for logical address 10 is : (6 * 4 bytes) + 3 byte offset = 27

Paging

- **A user program views memory as a single contiguous memory space**
- **In actuality, the user program is scattered throughout physical memory in page sized chunks**
- **Since the operating system is responsible for managing memory, it must be aware of the allocation details of the physical memory**
  - which frames are allocated
    - and to which page of which process each frame is allocated
  - which frames are available
  - how many frames there are in total

Allocating Frames to a New Process



Before allocation

After allocation

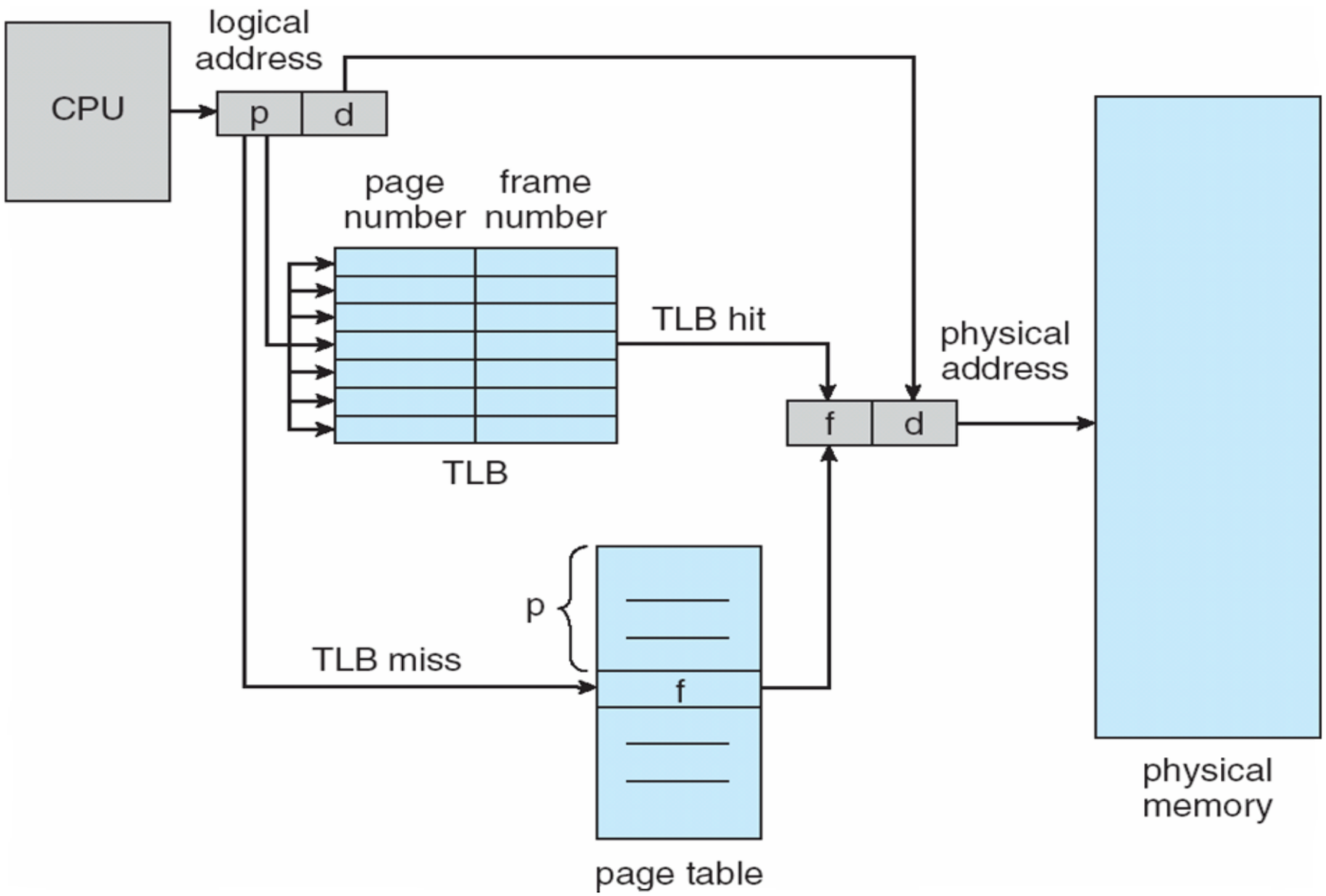
Implementation of Page Table (The Slow Way)

- **Page table is kept in main memory**
- **Page-table base register (PTBR) points to the page table in memory**
  - Another register that is loaded when process is dispatched to run
- **In this scheme every data/instruction access requires two memory accesses**
  - One for the page table and one for the data / instruction

Implementation of Page Table (The Better Way)

- **The two memory access problem can be solved by the use of a special fast-lookup hardware cache of associative memory called translation look-aside buffers (TLBs)**
  - Lookup page number in TLB to quickly determine the frame number
- **TLBs are typically small (64 to 1,024 entries)**
- **On a TLB miss, value is loaded into the TLB for faster access next time**
  - Replacement policies must be considered (which entries to overwrite when TLB is full)
  - Some entries can be wired down for permanent fast access (entries for kernel code)

Paging Hardware with TLB



Memory Protection

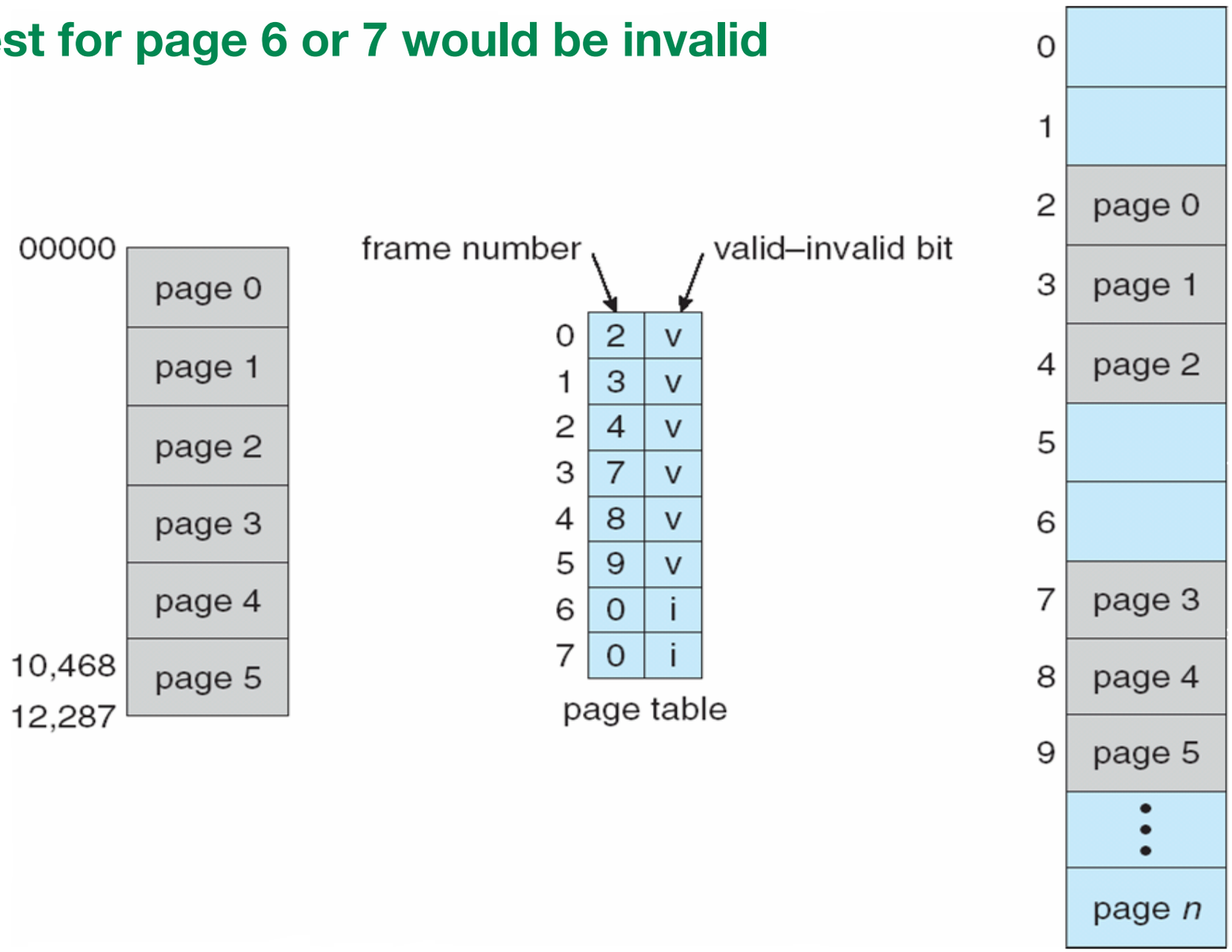
- **In a paged environment memory protection is implemented by associating a protection bit with each frame**
  - Indicates if read-only or read-write access is allowed
  - Can also add more bits to indicate page execute-only, and so on

- **Valid-invalid bit attached to each entry in the page table**
  - “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page
  - “invalid” indicates that the page is not in the process’ logical address space

- **Any violations result in a trap to the kernel**

Valid (v) or Invalid (i) Bit in a Page Table

- **Pages 0 through 5 access valid frames in physical memory**
- **A page request for page 6 or 7 would be invalid**



Shared Pages

- **If multiple process are sharing common code**
  - Use only one copy of read-only (reentrant) code among those processes (i.e., text editors, compilers, window systems)
  - For portions of the process that are not shared, each can have it own pages

Shared Pages Example